Aalto-yliopisto
Insinööritieteiden
korkeakoulu

Jyry Hyvärinen

Online performance control of electric vehicle

Master's thesis, which has been given in for a degree of Master of Science in Technology.

Espoo 01.03.2017
Supervisor: Professor Kari Tammi
Instructor: Jari Vepsäläinen

Aalto-yliopisto
Insinööritieteiden korkeakoulu

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi

Diplomityön tiivistelmä

Tekijä Jyry Hyvärinen

Työn nimi Sähköajoneuvon suorituskyvyn hallinta internetin ylitse

Koulutusohjelma Konetekniikan koulutusohjelma

Pääaine Koneensuunnittelu **Koodi** K3001

Työn valvoja Prof. Kari Tammi

Työn ohjaaja(t) DI Jari Vepsäläinen

Päivämäärä 01.03.2017 **Sivumäärä** 39 + 9 **Kieli** englanti

Tiivistelmä

Kiinnostus esineiden internetiä (engl. Internet of Things, IoT) kohtaan kasvaa jatkuvasti. Vastauksena kasvavaan kysyntään opetuslaitosten on kehitettävä internetiin kytkettävien mekatronisten laitteiden opetusta.

Tämän tutkimuksen tarkoitus on parantaa internet teknologioiden opetusta sekä kehittää opetuksessa käytettäviä laitteita, joita käytetään mekatroniikan opetuksessa Aalto-yliopistossa. Päämäärä saavutetaan suunnittelemalla ja kehittämällä IoT-opetusalusta mekatroniikan opetukseen, jolla voidaan tarkkailla ja ohjata osana alustaa olevaa laitetta. Kehitetty alusta koostuu sähköajoneuvosta, laitteistosta internet kytkennän ja palvelimen kehittämiseen sekä tarvittavasta ohjelmistosta. Järjestelmän kehitys on jaettu kolmeen eri vaiheeseen. Ensimmäinen vaihe on luoda CAN-väylä datan siirtämiseksi ajoneuvon ja palvelintietokoneen välillä. Toinen vaihe on kehittää ohjelmisto, jolla hallitaan ja tulkataan CAN-väylän välityksellä siirrettävää dataa. Viimeinen kehitysvaihe on luoda yhteys palvelimen ja asiakastietokoneiden välille, jotta järjestelmän käyttäjät voivat hallita ja tarkkailla ajoneuvoa internetin välityksellä.

Kehitysprosessin tuloksena saatiin IoT-opetusalusta sähköajoneuvon tarkkailuun ja hallintaan, jota voidaan hyödyntää niin opetuksessa kuin tutkimuksessakin. Lisäksi tuloksissa esitellään järjestelmän testaamisesta saatuun dataan perustuvia opetusmenetelmiä, joiden toteutuksessa hyödynnetään ajoneuvon ja kehitetyn internetsovelluksen luomaa IoT-alustaa.

Avainsanat IoT, esineiden internet, mekatroniikka, opetus

Aalto University
School of Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi

Abstract of master's thesis

Author Jyry Hyvärinen	
Title of thesis Online performance control of electric vehicle	
Degree programme Programme of machine design	
Major Machine design	**Code** K3001
Thesis supervisor Prof. Kari Tammi	
Thesis advisor(s) M.Sc. Jari Vepsäläinen	
Date 01.03.2017 **Number of pages** 39 + 9	**Language** English

Abstract

There is a rising interest towards the internet of things (IoT). To keep up with the demand of this technology, it is important for universities to improve the teaching of mechatronics associated with internet.

The objective of the study is to improve the teaching of the internet technologies and the equipment utilized in education of mechatronics in Aalto University. This aim is achieved by developing an IoT mechatronics education platform that can be used to monitor and to control a running process. The platform consist of an electric vehicle, added hardware for data acquisition and internet capability implementation and the needed software. The development of the system is performed in three phases. The first is to establish a CAN bus between the server and the vehicle. The second is to process and handle the driving data acquired with the CAN bus. The last phase is to enable a server-client communication over the internet. The developed platform is tested by analyzing the data acquired form the vehicle.

The outcome of the study is an online control and monitoring platform of an electric vehicle, which can be used for education and research. In addition, different types of teaching methods are suggested based on the performed data analysis.

Keywords IoT, Internet of Things, mechatronics, education

Preface

I would like to thank my supervisor, Professor D.Sc. Kari Tammi, and advisor, M.Sc. Jari Vepsäläinen, for guidance and the staff of A!OLE project for making this thesis possible. In addition, special thanks for technical support, M.Sc. Ville Klar.

Espoo 01.03.2017

Jyry Hyvärinen

Table of contents

Nomenclature

I	[A]	current
N	[RPM]	rotation speed
P	[W]	power
R	[Ω]	resistance
U	[V]	voltage
τ	[Nm]	torque
ω	[rad/sec]	angular velocity

Abbreviations

ACK	Acknowledgement
API	Application Programming Interface
A!OLE	Aalto Online Learning
BLDC	Brushless Direct Current
CAN	Controller Area Network
COB	Communication Object
CRC	Cyclical Recovery Checking
CSS	Cascading Style Sheets
DF	Data Frame
DLC	Data Length Code
ECU	Electronic Control Unit
EOF	End-of-Frame
EV	Electric Vehicle
EXT	Extended
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identifier
IDE	Identifier Extension
IFS	Interframe Space
I/O	Input/Output
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
MCU	Microcontroller
OD	Object Dictionary
OS	Operating System
OSI	Open System Interconnection
PDO	Process Data Object
RPDO	Receive Process Data Object
RPM	Revolutions Per Minute
RRF	Remote Request Frame
RTR	Remote Transmission Request
SDO	Service Data Object
SOF	Start of Frame
SPI	Serial Peripheral Interface
SRR	Substitute Remote Request
SSE	Server-Sent Event
STD	Standard
TCP	Transmission Control Protocol
TPDO	Transfer Process Data Object
URL	Uniform Resource Locator
WSGI	Web Server Gateway Interface

1 Introduction

1.1 Background

Mechatronics is a combination of mechanics, electronics and computer science. The basics of these fields and theories behind the phenomena are taught on lectures, calculation exercises and literature research exercises. It is commonly accepted that to teach undergraduate and graduate students to develop and design sophisticated mechatronic machines, the learnt theories and technologies should also be practiced with hands-on exercises and projects.

The internet of things (IoT) is a paradigm in which things and objects are connected to the internet and accessible online. Therefore, it is rapidly growing to be an important part of mechatronics. The presence of IoT raises a new range of challenges around the concepts which mechatronics is based on, thus affecting the ways mechatronics is seen and how it should be taught. It is suggested that the ways in which designers and educators in particular respond to these changes and manages the challenges will have a major impact on how the relationship between mechatronics and IoT develop over time.[1]

The rise of IoT among mechatronics makes it important for students to have access to devices, which utilize the possibilities of the internet. The machine design department of Aalto University, where mechatronics is taught and studied, does not have teaching devices with internet capability.

This thesis is part of the Aalto Online Learning, A!OLE, project. The aim of this project is to develop, explore and evaluate new types of technical solutions and pedagogical models for online/blended learning and disseminate recognized best methods and models into wider use in order to benefit students and teachers in Aalto University. Desired outcome of the project is to create a culture shift where digitalization of teaching is an asset to build on. This also includes discussing the critical point of view what is possibly lost in digitalization of teaching. Face-to-face contact between students and teachers is among the most valuable assets of university teaching and it should remain as one. Many daily tasks can be automated to save the time of teachers to create interactive and motivating learning content. [2]

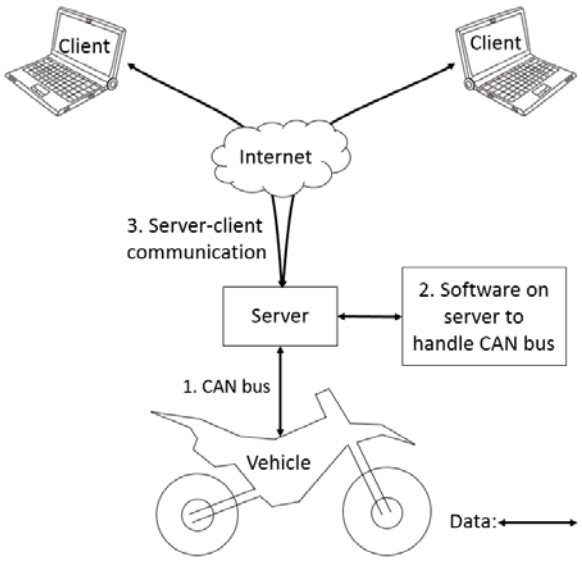

Figure 1 System data transfer topology.

1.2 Objective

The objective of this study is to support the aim of the A!OLE project in terms of developing and improving technical solutions to support learning and the teaching of mechatronic devices associated with the internet. In addition, pedagogical teaching models utilizing the developed technical solution are proposed.

The goal of this master's thesis is to design and to develop an open source online control platform for teaching and research. The primary goal is to acquire data from the vehicle and make it available for students and researchers online. This makes it possible to perform measurements remotely. The secondary goal is the possibility to program the controller of the vehicle and set control parameters with the application. As a result, an open source online electric vehicle monitoring and control platform is developed.

1.3 Scope

In Figure 1 the topology of data transfer in the system is presented. The first phase is to setup controller area network (CAN) bus between the server computer and the vehicle to enable data acquisition and communication. In the second phase the data is processed, i.e., parsed and converted in to the appropriate format. The third phase is to enable communication between the server and the clients, i.e., gathered data is transmitted to the client and user input can be used to manipulate the studied system.

The focus is on finding adequate solutions to the previously defined phases to enable communication between the client and the electric vehicle and implementing the solutions to the platform. The electric vehicle has been designed and developed previously. This process is outlined in detail, but the vehicle used is briefly described in those parts considered necessary and needed technical details are given. Access to the developed web application is restricted to be accessed only in the local network to avoid internet security problems. In addition, test drives on the road are not performed and the generated teaching methods are not tested with students for this thesis.

1.4 Structure of thesis

In chapter 2 the system topology shown in Figure 1 and the solutions found in literature are explained. In chapter 3 the utilized electric vehicle, hardware and software tools are presented. Testing procedure and methods to analyze the data are given in chapter 4. The developed software and created teaching method are presented in chapter 5. The last chapter discuss the outcome of the study, flaws of the developed system and further improvements.

2 Communication between vehicle and client

In this chapter the solutions found for product development phases in Figure 1 are presented and the theories of these technologies are studied. The first phase is to establish a serial bus between the server computer and the vehicle. The second phase is to process the data to the appropriate format and the last phase is to enable data transmission between server and client.

2.1 Data acquisition

The first phase is to enable communication between the server computer and the vehicle controller. The communication is performed with serial communication. It is a method to send data between devices, such as computers or devices inside computers one bit at a time sequentially over a communication channel (e.g. wire) or computer bus (wire, software and communication protocol). The serial communication architecture used in this thesis is CAN bus, therefore CAN standard and CANopen protocol are presented.

2.1.1 CAN bus

Controller area network (CAN) is an internationally standardized low level serial communication protocol [3], [4]. It was originally designed for the automotive industry in the 1980's but has become widely used in other industries and applications. It is mainly used in embedded systems and it provides fast communication among the nodes satisfying real time requirements. [5, p. 4]

The CAN protocol retains the following features:

- Connection – CAN bus allows multiple devices to be connected at the same time and there is no logical limit to the number of connectable units.
- Message transmission – all messages are transmitted in predefined format. When the bus is free, any of the connected nodes can start transmission. If multiple units transmit messages simultaneously, message ID defines the priority of the message and the transmission of the message with the lowest ID will be completed first [6, p. 45]
- Remote data request – Nodes can request data from other nodes by sending remote frame to the bus.
- Bus line flexibility – The nodes connected to the bus do not have specified addresses. Therefore, adding and removing units does not require changes in software or hardware.
- Error detection – All nodes can detect errors. If a faulty message is detected, the message transmission will be repeated.
- Connection speed – CAN supports multiple different connections speeds up to 1 Mbit/s. [7, p. 5]

2.1.1.1 Bus line

CAN bus allows different nodes (e.g. controllers, computers) in the system to communicate with each other without a host computer. In CAN bus all devices are connected to each other, thus all transmitted messages are passed to all nodes in the network. Each message has a unique identifier part which allows the devices in the network to determine whether the message is relevant or if it should be filtered. The identifier part of the message also includes the priority of the message. [8] The topology of CAN bus is presented in Figure 2.

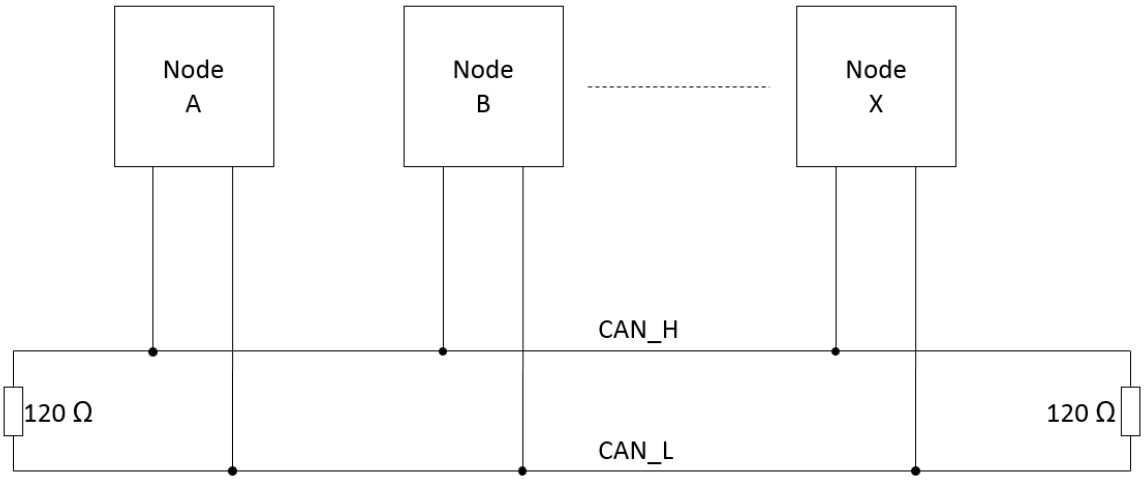

Figure 2 CAN bus topology [6].

All the nodes in the network are connected with two wires, CAN_H and CAN_L in Figure 2. The bus line is terminated by resistors which are essential to suppress electrical reflection on the bus. It is not advised to connect the resistors to a CAN node, but at ends of the bus. This is because the bus will lose proper termination, if this specific nope equipped with the resistor is removed from the line. Typically the nominal value of the resistor is 120 Ω [4]. [6, p. 133]

2.1.1.2 Frame types

CAN offers four different frame types:

- data frame (DF)
- remote request frame (RRF)
- error frame
- overload frame.

Data frame is the only frame type for actual data transmission. It transfers data from the transmitting node to the bus. After the transmission, the frame can be received by one or numerous nodes. There are two message formats. Standard format and extended format. The only differences between these two formats are amount of identifier bits and allowable messages. Standard format with 11 bit identifier allows a total of $2^{11} = 2048$ different messages while the extended format with 29 bit identifier allows for more than 536 million (2^{29}) different messages. Both, the standard and extended frame format can be used in the same CAN bus after the CAN 2.0B update. Standard, 11 bit message ID frames always have higher priority than the extended 29 but message ID frame with identical 11 bit base identifier. Bit-by-bit structure is presented in Table 1. [6, pp. 36, 53–54]

Remote frames are data frames without the data field. Remote frame requests another node to transmit a message. When the requested message is transmitted by the node, which is defined with ID, the message can be accepted by any of the other nodes in the bus line if they are configured to receive messages with the current ID. Once the node with the specified ID detection accepts the message, it transmits a response to the node which sent the remote request frame. Structure presented in Table 1. [6, p. 39]

Per definition a CAN data and remote frame has the following components:

- SOF (start of frame) – indicates the start of data and remote frames
- arbitration field, red – includes message ID and remote transmission request (RTR)
- control field, orange – includes data size and message ID length
- data field, light green – transmitted data (not in remote frame)
- CRC (cyclic recovery checking) field, dark green – error detection code
- ACK (acknowledgement) field, light blue – serves as confirmation of successful CRC
- EOF (End-of-frame), dark blue – terminates data or remote frame
- IFS (interframe space) – mandatory break between frames. [5, pp. 3–4], [6, pp. 43–56]

Colors in previous list show in which component each bit belong in Table 1.

Table 1 Standard and extended data frame structure.

Field name (standard)	**Field name (extended)**	**Length (bit)**	**Purpose**
Start of Frame	Start of frame	1	Indicates the start of frame transmission
ID	-	11	Message ID & priority
-	ID A	11	First part of message ID & priority
RTR	-	1	Message type (RRF/DF)
-	SRR	1	Substitute and placeholder for RTR in extended format
IDE bit	IDE bit	1	STD or EXT
-	ID B	18	Second part of message ID & priority
-	RTR	1	Message type (RRF/DF)
Reserved bit	-	1	For possible use by future standard amendment
-	Reserved bits	2	For possible use by future standard amendment
DLC	DLC	4	Number of bytes of data in the message
Data field	Data field	0-64	Up to 8 bytes of data. Not in remote frame.
CRC	CRC	15	Checksum of the preceding data for error detection
CRC delimiter	CRC delimiter	1	Stop CRC
ACK slot	ACK slot	1	Indicator of successful CRC
ACK delimiter	ACK delimiter	1	Stop ACK
EOF	EOF	7	Indicates end of message
IFS	IFS	3	Mandatory break between all types of frames

An error frame signals the detection of an error condition in a receiving or transmitting node. The error detection is accomplished by violating the formatting rules of a CAN message on purpose. The intended violation ensures the destruction of the faulty data or remote frame.

When an error is detected a error frame is sent by the CAN node which has detected the fault. The faulty message is then rejected by all node and retransmission of the faulty message starts. An error frame consists of a 6-12 bit error flag, an 8 bit error delimiter and a 3 bit IFS. [6, pp. 10–11], [9, pp. 57–58]

Overload frame is a special instance of an error frame. The distinctive feature is that overload frame does not restart the transmission of the previous frame but request a delay of the next transmission. Like any error frame it consists of a 6-12 bit overload flag and an 8 bit overload delimiter. The overload frame can only occur between data or remote frame transmission, right after the EOF of the last transmitted frame. [6, pp. 66–67]

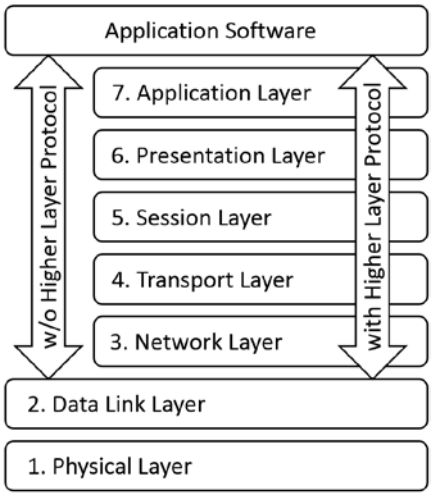

Figure 3 OSI 7 layer reference model. Adapted from [6, p. 22].

The open system interconnection (OSI) layer reference model, shown in Figure 3, specifies the seven levels from physical layer to the application layer. Physical layer represents the actual hardware i.e., physical connection between nodes, voltage levels and timing. The data link layer connects the actual data to the protocol in terms of sending, receiving and validating data. The application layer is the layer that actually interacts with the application of the CAN device or the OS (operating system). [6, pp. 22–23]

The connection between the data link layer and the application layer is bypassed in the standard CAN implementation to minimize memory resources. To cover the missing layer (3-6) additional software resources are required. These resources are provided in higher layer protocols, such as CANopen, DeviceNet and J1939. Using higher-level protocol simplifies software development and the developer can focus solely on programming the actual application layer. [6, pp. 22–23]

2.1.2 CANopen

CANopen is a CAN based internationally standardized communication system. It consists of high-layer protocols and profile specifications. CANopen has been developed as a standardized embedded network with highly flexible configuration capabilities. The main benefits of CANopen is that it releases the system developer from dealing with CAN hardware specific details and low-level layers. In addition it provides standardized communication objects (COB), configuration and network management data. CANopen makes communication between devices of different suppliers and manufacturers possible, and therefore enables interchangeability and eases product development. [10], [11]

According to the seven layer OSI model shown in Figure 3, CANopen is based on the data link layer according to ISO 11898-1 [3] and it assumes a physical layer according to ISO 11898-2 [4]which are covered by CAN. CANopen covers the higher layers: network, transport, session, presentation and application layers. The application layer describes how to configure, transfer data and synchronize CANopen devices. The application layer is defined in specification CiA DS 301 [12]. It includes the structure of the object dictionary (OD), network management and communication objects, such as transmit process data objects and receive process data objects. [7], [10]

One of the main features of CANopen is a table called object dictionary, in which configuration and process data is stored. A requirement for all CANopen devices is to implement an object dictionary. The CANopen standard defines that a certain address and address ranges in the OD must contain specific parameters. For example, it is defined in the standard that index 1008h with sub-index 00h must contain the product/device name. [13], [14]

CANopen devices utilize object dictionaries as a method of communication. For example, to initiate an event on a CANopen device, a massage is sent to change a value of an object in the object dictionary. Then the change is interpreted as a signal to start the event. The master node may also need to read the configuration and process data from the object dictionary. The first mechanism to access the OD is service data objects (SDOs) and the other is process data objects (PDOs). [13], [14]

The message format for a CANopen frame is based on the CAN frame format. The COB-ID, commonly referred to in CANopen, comprises of the ID and the control bits. In CANopen the 11 bit ID is split into two sections, a 4 bit function code and a 7 bit CANopen node ID. In addition, the message has DLC and the data field. The structure of data frame in CANopen is shown in Figure 4. [13]

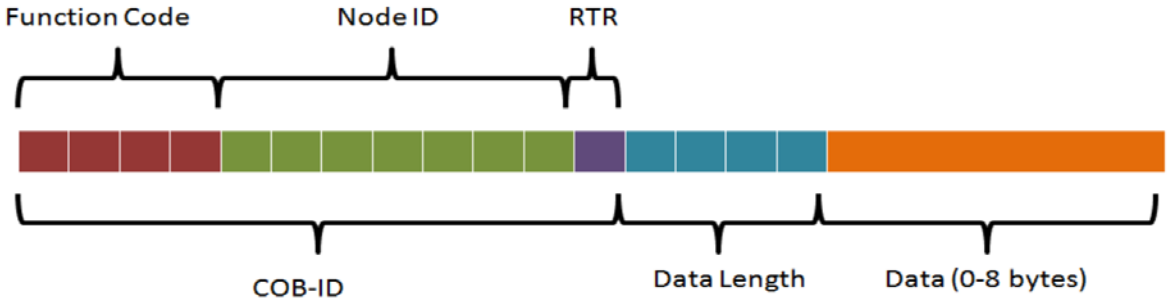

Figure 4 CANopen data frame format [13].

The main task of CANopen is to exchange process data. For this purpose most of the CAN identifiers and object dictionary entries are provided. CANopen offers two different ways to transfer data. The first method, service data objects (SDOs), is based on client-server communication and it allows direct access for the client node on the object dictionary of the server node using the indexes and sub-indexes. The other method, process data objects (PDOs), provides more efficient transmission of data. With PDOs the data transfer occurs according to the "producer-consumer principle" shown in Figure 5. This means that a message sent by a producer node can be received by all other nodes, referred to as consumers. [14], [15]

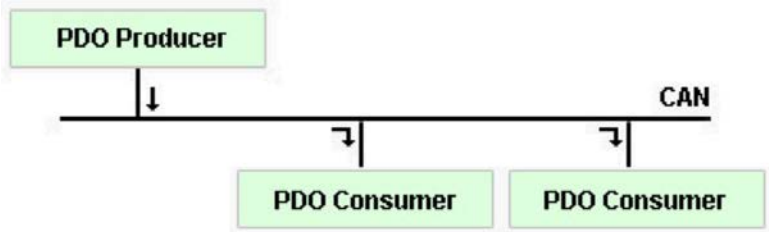

Figure 5 Producer-consumer principle.

The CANopen protocol specifies that each individual node in the bus line must setup a server that manages the read and write requests to the object dictionary. This method to directly access the object dictionary of another node is the SDO. The node accessing the OD of another node is the client that always starts the communication by sending a request frame to the server. SDOs are used for the configuration of devices and for the transmission of larger data blocks.[13]–[15]

SDOs are not a suitable solution to access the process data, stored in the OD, since CANopen protocol also has the requirement that a node must be able to send data itself without a request from CANopen master. To minimize the data exchange time and to avoid unnecessary traffic in the bus, data is sent with transmit process data objects (TPDOs). To transmit the desired data, e.g., motor rotation speed and supply current, these variables are bound to the TPDOs transmitting continuously. On the contrary, to make a node listen to specific PDOs in the bus, the listening node is programmed to receive process objects. On the consumer node side the received PDOs are referred as to RPDOs. [13], [15]

2.2 Web development

The third phase in the development process is to pass the acquired process data to the client and to allow the user to manipulate the process parameters online. To accomplish this phase a web application is developed. In this section, the basics of the web application development and internet technologies for data transfer are studied.

2.2.1 Web application

When speaking of web pages or applications it must be made clear, what they are and what are the differences. In general, it is considered that web applications allows the user to manipulate and save something, whereas a web page mostly provides information. Nevertheless, applications and pages are similar. Both are graphical presentations, for example showing logos, other identity information and offer vast amount of information. There are not clear instructions to define whether it is a web page or a web application, but the presence or absence of some functionalities and features are good indicators to determine which alternative is in question. Furthermore, there are four different types of web applications, daemonic programs, auxiliary applications, transient applications and sovereign applications. [16, pp. 2–4]

Applications that do not normally interact with users are referred as daemonic programs. They serve quietly and invisibly in the background and do not require continuous human interaction. A typical daemonic program would be a process that checks every 10 minutes for new alarms or events on the server. [17, pp. 103–116]

Auxiliary or parasite applications are continuously present, but only perform tasks that support the main application or page. For example, a parasite program may monitor how much system resources are used and how much memory is available. They are small and integrated to a greater whole. [17, pp. 103–116]

Transient applications do only one simple task. It appears when needed, fulfills the task, leaves and lets the user to continue normal activity. For example, a comparison chart of flight prices and a calendar pop-up button next to a date select input field on a webpage are transient applications. [17, pp. 103–116]

The fourth type of applications is a "sovereign" application. Sovereign application can be described as the only one on the screen, seizing the attention of the user for long periods of time. The user tends to keep the application running continuously. A good example of a sovereign application is a form filled by the user. [17, pp. 103–116]

A full-scale web program usually consists of all the previously mentioned application types, which all support the performed main task or tasks. As stated, the sovereign application is the one seizing the attention of the user for long periods of time. The rest of the applications improve the main application by adding functionalities and features.

Fowler introduces an eleven item table [16], which can be used to determine whether the developed system is an application, page or perhaps a hybrid. Considering the goals and development phases of this study, five out of eleven (5/11) items in the table indicate that the developed platform is an application.

Items indicating that the developed software is a web application:

1. relationship between user and application, task critical relationship between user and application
2. nature of the interaction: continuous data exchange and task-oriented model
3. frequency of use: constant use
4. response time: responsive machine control
5. interaction time: real-time data feed

From the remaining items on the list three out of eleven (3/11) indicate that the system is a hybrid and the rest (3/11) state that it would be a web page. Based on these items it can be said that the developed software is a web application.

Web applications offer great advantages but in comparison to more traditional desktop programs present some irritating disadvantages. The presented advantages and disadvantages are modified from Fowler [16, pp. 4–11] to serve and suite the purpose of this thesis.

Advantages of web application compared to desktop programs:

- no compilation
    - JavaScript (JS) and Hypertext Markup Language (HTML) do not require compilation and code is immediately ready for testing.
- not installation to individual computers
    - Facilities with great amount of people, e.g., universities and large corporations, do not need to install software to each computer.

- portable, even mobile
    - Users can access the application and data anywhere
- global
    - Instead of distributing copies of databases around the world, all data can be stored in a single central database.
- application update for all users at once.
    - Application updates are immediately available and initialized for all users.

The previous advantages are thanks to internet making applications accessible worldwide, but accessibility also has downsides. The disadvantages and threats created by internet are listed below.

Disadvantages and potential threats:

- **security risk**
  The security becomes a major issue, when the server is open to be accessed via internet. This exposes the server and system for undesired use by unauthorized people.
- **damage risk**
  The damage risk is a direct consequence of security problems. An unauthorized person may harm the system or load malware to the server, thus crippling the application.
- **network failure**
  If the network is incapacitated because of spam attacks, equipment breakdown, configuration mistakes or other problems, the running application is stopped and useless.
- **decreased performance**
  Large tables take long to load and checking inputs immediately causes major decrease in performance. Also, since the programs on the backend do not know what is happening on the front end, validation and error checking can be time consuming.
- **browser incompatibilities.**
  Browsers version and type might affect the functioning of the application.

2.2.2 HTTP

The Hypertext Transfer Protocol (HTTP) is a request-response protocol based on the transmission control protocol/internet protocol (TCP/IP), which is utilized by clients and servers to transfer data, for example images, data files and audio files. A diagram of the HTTP protocol mechanism is shown in Figure 6. The HTTP client opens a TCP connection to the server and sends a request. Once the server receives the request it checks the database for new data. If requested data is available, the server responds with the data, otherwise it will respond with a code, which indicates that no new data is available.

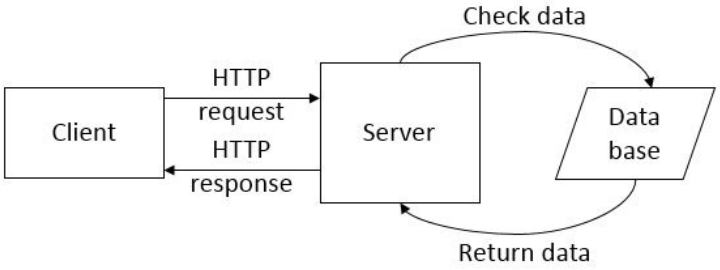

Figure 6 HTTP request/response diagram.

HTTP has three basic features:

- stateless
    - The server and the client are aware of each other only during the connection, i.e., when request is made. After the connection is closed, both parties forget each other. Due to this nature, the client and the browser are not able to retain information between requests across the web pages.
- connectionless
    - The client send a request to the server and once the request is performed, the client disconnects from the server and starts to wait for a response. The server processes the request and re-establishes the connection with the client to send a response.
- media independent.
    - Any kind of data can be transmitted by HTTP if both, the server and the client know how to handle the content. It is required by the server and the client to specify the content. [18, pp. 35–37]

HTTP defines different types of request methods to indicate the desired action to be performed for a given resource. Two of the most commonly used methods, and essential for this thesis are GET and POST methods. Both methods are used to request data from the server but there are situations where the other is never be used. The GET methods should be used only for data retrieving from the server because the request parameters are saved in the browser history and the data is visible to everyone in the universal resource locator (URL). On contrary, the POST method can be used to submit data to be processed on the server because the posted data is not visible for others and the parameters are not saved in the browser history. The main differences concerns internet security and for example GET method should never be used to transmit passwords and other sensitive information. In addition, the data type with GET method is restricted to ASCII characters whereas POST methods does not have restrictions and even binary data is allowed. There are seven other methods available but they are not utilized in the developed of this web application. [19], [20], [21]

2.2.3 TCP/IP

Instead of being single protocol the TCP/IP is a combination of multiple protocols used in internet communication. The central protocols included are transmission control protocol (TCP) and internet protocol (IP).

IP is a packet switching protocol. The main responsibility of this protocol in the internet connection is to deliver data packets from one host to another laying in the same or different network. This is achieved by adding a header that contains addressing and control information. The header contains a source and destination address that are defined as IP address. IP provides segmentation and reassembly of lengthy packets into smaller packets. This is useful, because packets can go through networks that have different rules for maximum packet length on the way to the destination network. This protocol treats each individual packet as an entity unrelated to any other packet. [18, p. 23]

TCP is a connection-oriented, end-to-end reliable protocol designed to fit into a layered hierarchy of protocols which support multi-network applications. This protocol establishes a connection between the source and the destination. During the connection, TCP breaks data into segments at the source and reassembles them at the destination. If a defected segment is received or the segment is not received at all, it will be resent. [18, pp. 29–33]

2.2.4 Server-Sent Events

In a standard HTTP request-response scenario a client opens a connection, sends a HTTP request to the server, receives a HTTP response back and then the server closes the connection once the response is fully handled. The client always initializes the communication on a data request. In contrast, Server-Sent Events (SSE) is a mechanism that allows the server to automatically push data to the client once the client-server connection is opened. When the connection is established, it is the server providing the data and deciding to send it to the client whenever new package of data is available. Once a new data event occurs on the server, the data event is sent to the client by the server. To fully understand SSE it is vital to also understand the technologies supporting the server-to-client communication, i.e., polling and long-polling. [22], [23]

Polling is a mechanism initialized by the client to repeatedly send new HTTP requests to the server. If the desired data is available on the server, it is transmitted to the client and the connection is closed. On the other hand, if the requested data is not available then the server sends appropriate indication and closes the connection. To get new data the client must send a new request. [23]

With long-polling, the client sends a HTTP request to the server. If the requested data is not ready or available, the server keeps the connection open until the data is available. Once the server acquires data for the client, the open connection is used to finish the request. After the data is transmitted to client the connection is closed. [23]

SSE is similar mechanism to long-polling, except it transmits multiple messages or data packages from the server to the client per connection. The request is initialized by the client, but once the connection is opened it keeps running. The server sends the data to the client while still keeping the connection open, ready to be used when new data is available. Once the new data is ready for transmission the same connection is utilized to push the data to the client with the same connection. The client processes the messages sent back from the server individually without closing the connection after each sent data package, also known as events. SSE defines a dedicated media type that describes the format of individual events sent by the server to the client. SSE also offers standard JavaScript client application programming interface (API) implemented to most modern browsers. [24], [25]

As described, SSE is a technology that enables constant data stream of events, noticed by the server to the client. It offers a half-duplex, one directional, connection solution from server to client. The half-duplex nature of SSE is the major drawback of this technology.

3 Platform development

The goal of this thesis is to develop an open source online control application which allows user to monitor and manipulate a running process. The controlled process in thesis is a powertrain of a light weight electric vehicle. This type of controlled device sets certain challenges, such as third party software restriction for setting certain process parameters and the electric vehicle has limited installation space for new devices. In the first section the requirements to fulfill the desired features are discussed. In the second section the specifications of the electric vehicle, i.e., the test machine, are presented. The last two sections discuss the hardware added to the electric vehicle for this thesis and the used software tools.

3.1 Requirements

The main goal is to develop an online electric vehicle control platform to improve online teaching and learning. To achieve the goal the features and specifications must be set considering the usefulness of the features in teaching. This section presents the desired features and specifications of the platform.

The wanted features for the developed online control application are

1. driving data from vehicle to user (primary goal)
2. user can adjust the running system (secondary goal)
3. save data (additional goal)
4. stored data to user, i.e., student or researcher (additional goal).

The requirement list on the next page in Table 2 is redrawn according to the previous features. The requirements described as demands (D) are the functionalities, which need to be fulfilled to meet the primary goal. Other requirements to fulfill secondary and additional goals are described as wishes (W).

Table 2 Requirement specification. Demands (D) and wishes (W).

Requirement:	Priority:
Geometry	
- Server/data processing computer must fit to the EV: small (max. 10x10x5 cm^3)	D
Data	
- Process data gathering (5 data obj.)	D
- Process parameter setting (1 param.)	W
CAN bus communication	
- Configure ECU TPDOs	D
- Configure ECU RPDOs	W
- Establish CAN bus between server and vehicle CAN system	D
-Data sampling rate (20 Hz)	W
Application backend	
- Receive CAN messages (ECU → server)	D
- Process raw data on CAN messages	D
- Set up web server	D
- Create web application	D
- Transmit processed raw data to web application (server → client)	D
- Receive client data input (client → server)	W
- Convert client input data to CAN format	W
- Send converted input data (server → ECU)	W
- Save processed raw data to database	W
- Upload file form database to client	W
Application frontend	
- Display processed raw data to user (running values)	D
- User data input fields (user → client)	W
- Data saving initialization	W
- Data file download link	W
- Visual presentation of data (graphs)	W
- Clear user interface	W

Processing the raw data includes solving how data is bound to PDOs and how the data needs to be parsed and converted to gain correct engineering unit values. After the data is processed in the correct manner, it can be passed further in the software, eventually ending up to the client and displayed as an engineering unit value for the user.

The gathered process data objects are supply voltage, supply current, motor rotation speed, motor torque and throttle input voltage. Based on these variables, efficiency calculations over the power train can be performed and the data can be used to verify simulation of the vehicle. For example, simulating the response time of the motor speed to throttle input voltage.

3.2 Electric vehicle specifications

In this section the electric vehicle is presented briefly. The controller, motor and power supply are the most important components of the vehicle and supply most of the desired data, therefore they are described in more detail. In addition, a throttle potentiometer on the vehicle is used to generate command signal. A schematic diagram of the electric vehicle and added hardware, i.e., the developed platform, is shown in Figure 7.

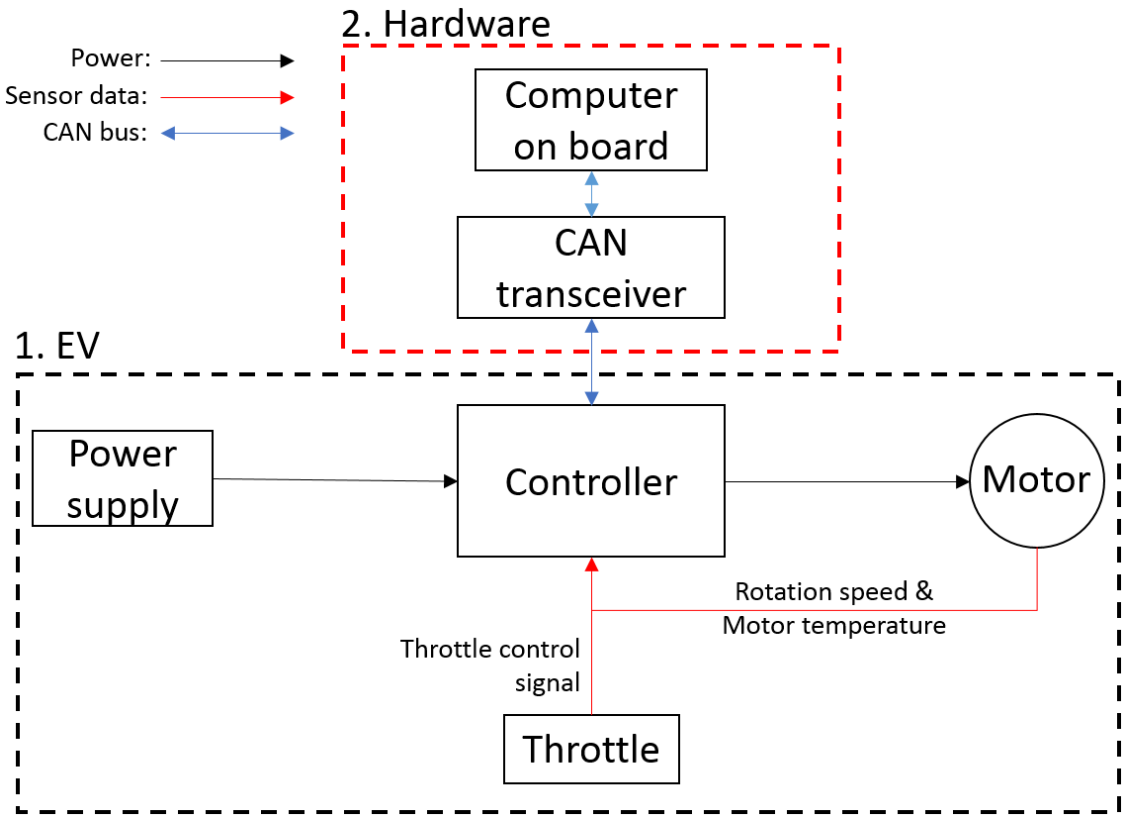

Figure 7 Vehicle and hardware diagram.

The electric vehicle was developed on the base of a Derbi Senda R X-treme 2006 model. Excessive parts, such as the engine, fuel tank and radiator were removed from the original design assembly. The frame, wheels and brakes remained original, although minor changes were made to fit the new electric powertrain components.

3.2.1 Controller

The electronic control unit (ECU) is an embedded electronic device and in theory, it is a computer, which handles signals from sensors in the controlled system. Depending on the received data from the sensors, the ECU controls the output of the power source and the motor. In addition, the ECU keeps track of the performance of the controlled system and regulates the provided energy to the actuators. Typically, ECUs have lots of integrated characteristics, i.e. analog and digital I/O, power device control and communication protocol (CANopen). [26]

Table 3 Input voltages and output current ratings for Sevcon Gen4 size 2.

Nominal operating voltage (V):	48
Conventional working voltage range (V):	25,2 – 57,6
Working voltage limits (V):	19,3 – 69,9
Non-operational overvoltage limit (V):	79,2
Short term (2 min.) current rating (A):	275
Continuous (1 h) current rating (A):	110

The used ECU model in the electric vehicle is a Gen4 size 2 of Sevcon designed for a nominal operating voltage of 48 V. More detailed electrical operation characteristics are presented in Table 3. The controller utilizes CANopen for communication [27]. All of the used measurement data is acquired from the ECU by CAN bus.

3.2.2 Motor

The used motor in the vehicle is a 10 kW 96 V brushless direct current (BLDC) motor manufactured by Golden Motor. The motor has a built in absolute UVW encoder and temperature sensor, which are observed by the ECU. When using the correct operating voltage, the motor is capable of providing maximum torque of 32,6 Nm and rotational speed of 4700 RPM. [28], [29]

3.2.3 Power supply

For research and development convenience, the batteries of the EV were removed. Instead, an external power supply was used to power the vehicle. The characteristics of the power supply are in Table 4.

Table 4 Power supply characteristics.

Max. voltage:	70 V
Max. current:	4 A
Max. power output:	280 W

The used external power supply is not capable to provide enough power to accelerate the motor rotation speed to higher than 1000 RPM, thus the performed measurements are restricted. In addition, if the current demand of the ECU is higher than the power supply can supply the controller falls into fault state.

3.3 Hardware

In this section the computer used on board and the CAN transceiver in Figure 7 are presented. The requirements for the hardware are CAN bus capability, data processing and the ability to setup a web server.

3.3.1 Computer on board

For the web server setup, processing the CANopen communication and transmitting the processed data to the client, a computer is required. To accomplish these tasks a Raspberry Pi 3 model B was selected. In addition to full filling these requirements, the Raspberry Pi is cheap, compact and light, thus easy to install to the light weight EV. The use of Raspberry Pi enables further development, e.g., adding a touch display to the vehicle.

Raspberry Pi is a credit card-sized computer running on a Linux based OS, developed by the Raspberry Pi Foundation. It has USB and HDMI ports to enable monitor and keybord/mouse connection, and it can be used for the same purposes as a regular computer. Raspberry Pi is widely used to promote teaching of programming and computer science. [30], [31] Raspberry Pi has a wide user community, and a great amount of technical support is available for software development.

Figure 8 Raspberry Pi 3 model B.

Some features of Raspberry 3 B that make it an adequate solution for this system:

- multiple languages pre-installed (Python, C, C++, Java, Ruby)
- built-in wireless network chip
- run multiple programs simultaneously
- internal memory: 1 Gb
- storage memory: 8 – 32 Gb (Micro SD card)
- processor: 1,2 GHz quad-core ARM. [32]

Raspberry Pi can run on multiple different operating systems, such as, Windows 10 IoT Core, Ubuntu Mate and Debian-based Raspbian. Most of the operating systems are Linux-based because Linux is a free open source operating system. For this application, Raspbian Jessie with Pixel (Linux kernel release: 4.4.32-v7+, date: 15.11.2016) was installed on the Raspberry Pi.

3.3.2 CAN bus capability

The CANopen communication between the vehicle and the server computer is established with a PiCAN2 board manufactured by SK Pang Electronics Ltd. This board is cost-effective and easy-to-use, open source solution to implement CAN communication to Raspberry Pi. SK Pang provides instructions to install the needed software to send and receive CAN bus messages. The PiCAN2 board is popular among hobbyist, which increases the available support.

PiCAN2 board provides CAN bus capability for Raspberry Pi. This board is compatible with Raspberry Pi 2 and 3. It utilizes MCP2515 CAN controller microchip with a MCP2551 transceiver. [33]

The MCP2515 is as stand-alone CAN controller that implements the CAN 2.0B specifications. The chip has the capability to transmit and receive standard and extended data frames and remote frames. It interfaces with the microcontrollers (MCUs) via an industry standard serial peripheral interface (SPI). MCP2515 simplifies the applications that require interfacing with CAN bus. [34]

The MCP2551 is a high-speed fault-tolerant CAN device. It serves as an interface between a CAN protocol controller (MCP2515) and the physical bus. The MCP2551 chip provides differential transmit and receive capabilities to the CAN protocol controller, and it operates with communication speeds up to 1 Mb/s. [35]

The PiCAN2 board offers the following features:

- CAN v2.0B communication up to 1 Mb/s
- high speed SPI at 10 MHz
- standard and extended data frames
- remote frames
- 120 Ω terminator resistor pre-installed
- standard DB9 or screw terminal connection
- solder bridge to support different configurations of DB9 connector
- socketCAN driver. [33]

SPI is a synchronous serial data protocol used by microcontrollers to communicate with other microcontrollers or peripheral devices on short distances, mostly in embedded systems. [36]

The PiCAN 2 board is powered by the attached Raspberry Pi. The board also offers the possibility to power up the assembly via the DB9 connector or a 3-way screw terminal. The board is installed as shown in Figure 9. It is installed by aligning the connection strip of PiCAN 2 to the 40-way connector on a compatible Raspberry Pi. Spacers and screws and used to secure the board. [33]

Figure 9 PiCAN 2 board installed on top of Raspberry Pi 3.

Once the PiCAN2 board is installed, programming can be done in C or Python. SK Pang Electronics Ltd provides libraries and modules for both languages. Python is used to develop the backend software of the control platform. This minimize the programming languages thus reducing the complexity of the system, because server setup is fulfilled with Flask module on Python.

3.4 Software

In this section the software used, the software tools and the changes to the existing software on the ECU on the EV are presented. The aim was to keep the system simple and easy-to-understand. This means that the used amount of different programming languages should be kept at minimum. By keeping, the system simple and easy to read, it is possible for an inexperienced programmer to understand the basics of the IoT-device development.

3.4.1 ECU software changes

In order to change the software on the Sevcon Gen4 controller Design Verification Test (DVT) software is required. This software functions only with IXXAT Model – 1.01.0087.10200 (with galvanic isolation) USB to CAN adapter. To make changes to the program on the ECU, the controller must be on preoperational mode. This mode disables the controller so that the motor will not run while adjustments are being made. When the pre-operational mode is activated, the line contactors drop out. [37]

The ECU was configured to transmit motor rotation speed, motor current, motor torque, supply voltage, supply current, inverter temperature and throttle input voltage as TPDOs. More detailed information is Table 5. In addition, the controller was configured to receive the maximum forward rotation speed as RPDO.

Table 5 Transmitted data. Scaling factor (SF).

TPDO:	COB-ID:	Bytes:	Bits:	Index:	Subindex:	Name & unit:	SF:	Explanation:
1	0x100	1-4	0-31	0x606C	0	Speed [RPM]	1	Motor RPM
1	0x100	5-6	32-47	0x5100	1	Battery U [V]	0,0625	Supply U
1	0x100	7-8	48-63	0x5100	2	Battery I [A]	0,0625	Supply I
2	0x200	1-2	0-15	0X6078	0	Motor I [A]	1	Actual motor I
2	0x200	3	16-23	0x5100	4	T [°C]	1	Inverter temp.
2	0x200	4-5	24-39	0x6077	0	Torque [% of peak]	0,1	Motor torque
2	0x200	6-7	40-55	0x2220	0	Throttle U [V]	3,90625e-3	Control U
2	0x200	8	56-63	X	X	Empty	X	Empty byte

As shown in Figure 10, the rotation speed (velocity), supply voltage (battery voltage), supply current (battery current) are bound to the TPDO 1 and motor current, heatsink temperature (inverter temp.), torque (torque estimate) and throttle input voltage are bound to TPDO 2. The first bound data in the TPDO 1 is the rotation speed of the motor (velocity), which fills the first 32 bits (4 bytes) of the available 64 bits (8 bytes). This information is crucial in message parsing. In addition, when converting the values from hexadecimal format to decimal format, the byte order must be reversed and DVT software provides the scaling factors needed to scale the values to the correct engineering units. Essential information to process CAN messages is given in Table 5. Example data transmitted in TPDO 1 is presented in Table 6.

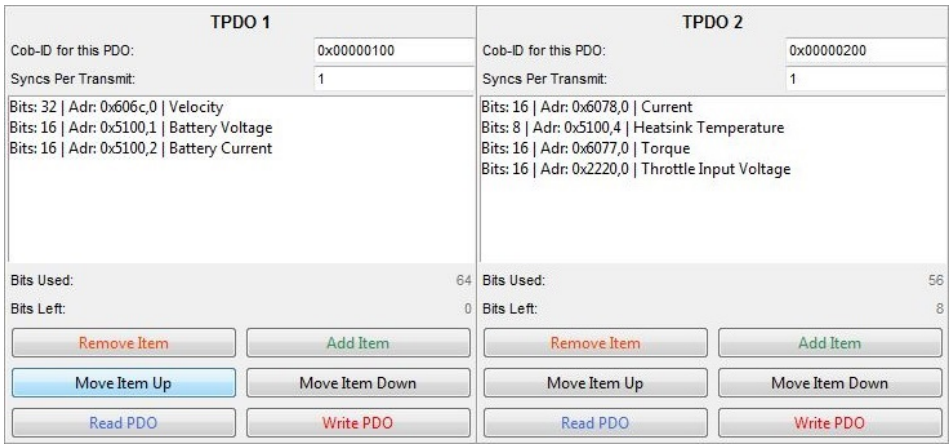

Figure 10 TPDO configuration menus.

For example, the conversion of the battery voltage in Table 6 to engineering units is performed by adding the value of byte 6 to the value of byte 5. Byte 6 is a multiply of 255, thus the hexadecimal value in byte 6 is 3*255 (=765) and the remainder value is in carried in byte 5. The value in decimal format is 765 + 1 = 766. Multiplying this value with the scaling factor of battery voltage (0,0625) results in a battery voltage value of 48 V.

Table 6 Example data of TPDO1.

Velocity				Battery Voltage		Battery Current	
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
00	00	00	00	01	03	00	00

In the PDO configuration menus, the COB-ID of each PDO can be defined. This is the ID, which is used to identify the message in the bus. The naming has two restrictions. The first one is inherent for CAN bus, thus two messages cannot have the same COB-ID. The second restriction is that the first digit of the ID cannot be 8, e.g., 0x8…, because this disables the message.

RPDO configuration on the electronic controller unit is needed to enable controlling of the ECU with CAN messages. The maximum rotation speed in forward direction is the only parameter, which is configured with CAN message. The RPDO configuration menu is shown in Figure 11. In the menu the COB-ID of the message and the parameter to receive the data is defined.

Figure 11 RPDO 1 configuration menu.

The size of the RPDO must be a multiply of 8. If the size of the configured data set is something else, dummy booleans need to be added to make the size correct. In the case of Figure 11 the dummy booleans are not needed because the size is 16 bits (2 bytes). The configured RPDO must be received by the ECU within five seconds after powering on. If the parameter is not received, an I/O initialization fault will occur and the controller falls into error mode and the system cannot be used.

After the wanted data is bound to the PDOs, the changes are confirmed with the Write PDO –button. The operational mode must be selected after settings changes are made and confirmed. The controller will not run the motor until the operational mode is selected. When the operational mode is activated the line contactors close.

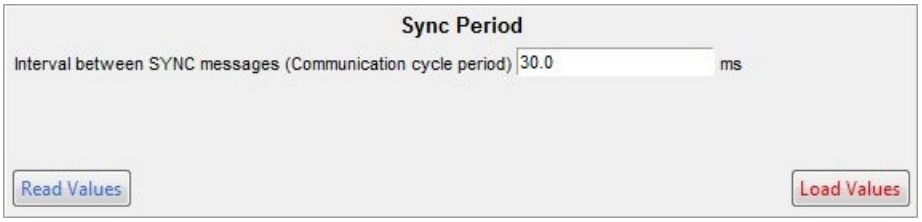

Figure 12 Synchronization period menu.

The interval between synchronization messages can be adjusted in a menu show in Figure 12. For testing and development the synchronization period was set to the value of 30 ms to meet the requirement in Table 2. If the synchronization period is toolow, the serial bus gets too busy and the developed software does not have enough time to process low priority messages before the next synchronization message, therefore all TPDOs are not handled. To

ensure a constant and continuous data stream without breaks, the frequency must be low enough. Once the developed software is ready, the synchronization period is adjusted to provide data at highest possible rate.

3.4.2 Programming languages

3.4.2.1 Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. The syntax of Python is considered to be easy to learn, emphasizing readability. As a language Python supports modules and packages, which encourage program modularity and code reuse. It is widely used in multiple fields in different types of applications. [38], [39, pp. 1–2]

Interpreted programming languages do not require compilation, thus they can be executed immediately with the interpreter. This way the developed program can be tested quickly in short cycles. However, interpreted languages have disadvantages, such as not being as efficient as languages that require compilation. [38]

Python is high-level in data structures, combined with dynamic typing and dynamic binding makes it attractive for rapid application development, as well as for use as a scripting and binding language to combine existing components. [38]

Simple syntax, easy readability and fast prototyping and testing makes Python well suited for this application. In addition, Python offers many tools for web development, such, as framework Django and micro frameworks Flask and Bottle. The standard libraries of Python support many internet protocols, e.g., HTML, JavaScript object notation (JSON) and socket interface. [40] In the developed web application, the backend software development (i.e., server setup and CAN communication) has been carried out with Python and Python based micro framework Flask.

3.4.2.2 JavaScript and jQuery

JavaScript (JS) is a lightweight, interpreted, high-level programming language with first-class function. It is best known as a scripting language for web pages, but it can be used for non-browser environments as well. JS allows web developers to implement functionalities on the occurrence of an event and complex feature to web applications and web pages. For example, most button initialized events, changing pictures, running values and input fields on web pages utilize JavaScript. [41], [42, pp. 1–2]

First-class functions means that the language treats functions as common variables, thus functions can be passed as arguments to other function and a function can be used as a return value of another function. First-class functions are a necessity for functional programming, in which the high-order functions are common. [43]

The appearance of web pages and applications are determined on the Hypertext Markup language (HTML) and Cascadian Style Sheet (CSS) files. JavaScript can add functionalities and change the appearance by manipulating these files, which are used to determine the content and appearance.

To make JS easier and more approachable, web developers can utilize jQuery, which is a small and fast JavaScript library with vast features. It makes the implementation of JS functionalities to HTML easier. JQuery has easy and short methods to find, change, add or delete elements in the HTML file. [44], [42, pp. 3–4]

On the contrary to Python, JS is mostly used on frontend development of web applications and pages and it runs, not on the server side like Python, but on the client side of the web. The developed events, which are initialized by button clicks and running data values, are implemented to the application with JS and jQuery.

3.4.2.3 HTML & CSS

HTML is a standardized markup language that is used to structure and outline the content of web pages and applications. Along with CSS and JavaScript it serves as a cornerstone technology for web design. Web browsers receive the HTML documents from the web servers with HTTP requests. Once a browser has received a HTML file, it renders the document and displays the content. HTML describes the appearance and semantics of web pages and applications. [45]

HTML utilizes elements as building blocks of the page. The elements are defined with tags and different types of tags can be used to create various types of elements for different purposes. The elements can contain attributes, which of most are name-pair values. By utilizing the attributes, elements can be referred to for information fetching or setting. For example, the *id* attribute is commonly used to name the element, thus by searching an element with certain *id*, information can be set to the element. [45], [46]

CSS is a language of style rules that are used to apply styling rules to the HTML content. For example, with CSS, colors and fonts can be changed and the page can be divided to multiple columns instead of just being one big canvas. To simplify the application appearance design, the rules are usually written into a separate .css file. This enables the possibility to display the HTML file with different styles without the need to make changes to the HTML file. This is particularly useful nowadays, because web application and pages can be browsed with various kind of devices that support different aspect ratios. [47], [48]

3.4.3 CAN bus tools

In order to establish CAN communication with the PiCAN2 board on Raspberry Pi, changes are required in the *config.txt* file, which is a file read by the Graphics Processing Unit (GPU) before initializing the Advanced RISC Machines (ARM) core. The information presented here is also available in the documentation of PiCAN2 board provider, SK Pang Electronics. [49]

The *config.txt* file can be opened with the following code on the command line of the Raspberry Pi:

- sudo nano /boot/config.txt

The first word “sudo” (super user do!) runs the following command with elevated privileges, which are required to perform certain administrative tasks, such as, editing the *config.txt* file. The second word “nano” is the name of the used command line text editor program, which

will be used to open the file. The last part of the command "/boot/confix.txt" indicated the path to the opened file.

The following three lines are added to the end of *config.txt* file:

1. dtparam=spi=on
2. dtoverlay=mcp2515-can0-overlay,oscillator=16000000,interrupt=25
3. dtoverlay=spi-bcm2835-overlay

The first and third lines enable SPI on Raspberry Pi. This is required, because PiCAN2 board and Raspberry Pi utilizes this bus to communicate. The second line attaches the previously mentioned MCP2515 to channel can0, adds oscillator to provide real-time capability and declares pin number 25 on Raspberry Pi as an interrupt pin.

The following command brings up the defined can0 interface for usage:

- sudo /sbin/ip link set can0 type can bitrate 1000000

This command sets the can0 channel to listen for CAN-bus communication at the desired bitrate. In this case, the bitrate is 1000000 bit/s (1 Mbit/s).

To utilize the established CAN bus on the Raspberry Pi, SocketCAN library is downloaded and installed. This library enables the capability to control the CAN bus and eases the needed development because some of the features of this application are included. The included features are listening to the CAN bus, arbitration, compiling the binary presentation of the received message, parsing the message to readable format and creating a buffer. The buffer ensures that the received data is uniform and messages are passed further to the program in the correct order. The library is available on bitbucket.org [50].

After installing the SocketCAN library, the establishment of CAN bus capability can be tested with a short Python script.

```
1 import can
2 bus = can.interface.Bus(channel="can0", bustype="socketcan_native")
3 notifier = can.Notifier(bus, [can.Printer()])
```

Figure 13 CAN bus listener.

The script in Figure 13 prints the next values to the Python interpreter: timestamp, COB-ID, flags, data length and the data in the message, 0-8 bytes. The flags indicate if the received message is error frame, extended frame format or RTR.

```
1 msg = can.Message(arbitration_id=0x200, data=[byte1, byte2, …, byte8]
```

Figure 14 Send message to the CAN bus.

To send data to the CAN bus, the Python script in Figure 14 is used. In order to utilize the code the SocketCAN library must be imported and the bus must be defined similarly as in Figure 14 on lines 1 and 2. The *arbitration_id* is the COB-ID of the RPDO listening for the message on the ECU.

As an example usage of the previous command, a message to program the maximum forward rotation speed of the motor is used. When setting the RPDOs to listen to, the ECU provides the size of the accepted data. In the example case, the message size is 16 bits, i.e., two bytes. If the maximum rotation speed is the first or the only data on the RPDO the data is sent on the two first bytes. When no longer needed, the can0 interface can be brought down with the following command:

- sudo /sbin/ip link set can0 down.

3.4.4 Web development framework

The web application was developed with Flask. It is a micro framework for Python, based on Werkzeug [51] toolkit and Jinja2 [52] template engine. Flask is considered as a micro framework because it does not require additional tools or libraries to be used. [53] Flask contains the following features:

- development server & debugger
- unit testing support
- RESTful request dispatching
- Jinja2 templates
- support for secure cookies
- WSGI 1.0 compliant
- Unicode based. [53]

Jinja2 is a modern and designer-friendly templating language for Python and it is modelled after the templates of Django, which is another popular framework for Python. Werkzeug is a utility library for the Web Server Gateway Interface (WSGI) [54], which itself is a protocol that ensures that web applications and webservers communicate successfully. [51], [52], [55]

4 Platform testing

To find problems and suitable teaching methods the developed system, i.e., the combination of the vehicle and the developed application is tested. This chapter presents the test parameters and testing procedure used to find the inconsistencies in the system.

As discussed in the section 3.2.3 the ECU falls into fault tolerance mode if the current demand is greater than the maximum supply current of the power supply. This restricts the maximum rotation speed of the motor. The test parameters are shown in Table 7.

Table 7 Test parameters.

Minimum speed:	100 RPM
Maximum speed:	1000 RPM
Increment:	50 RPM
Number of tests:	19
Measurement time:	5 s
Sampling rate:	30 ms
Values per test:	167

The following test procedure was generated to be used in the teaching models, but the increment and number of tests are not provided, thus creating variance on the measurements and results of each group or student. The same procedure was used to gather data for this thesis.

1. set maximum rotation speed
    a. first set maximum speed 100 RPM
2. accelerate the motor to maximum speed
    a. manual throttle used
    b. throttle input voltage 5,5 V
3. initiate data record
    a. begin once the RPM value does not increase
4. download data file
5. repeat until the whole speed range (100-1000 RPM) tested.

4.1 Data analysis

Based on the data received the efficiency of the powertrain and the accuracy of the set maximum speed are analyzed. This section presents how the acquired data was treated in order to analyze these physical units.

From each of the recorded data files the average of the provided rotation speeds was calculated. The resulting average values for each of the 19 different maximum speed limits were plotted to visualize the data and to generate an equation, which can be used to correct a possible offset between the set maximum speed and the measured data. The average values of each measurement, created graphs and generated equations are given in the Results 5.2.

The efficiency of the powertrain was analyzed with the data provided by the 500 RPM maximum rotation speed test. To calculate the efficiency during the 5 second sampling time, the averages of the supply voltage, the supply current, the motor rotation speed and the torque estimate were calculated. With these units, the average efficiency of the vehicle powertrain from battery to motor can be estimated. It is assumed that the powertrain ends to the motor,

therefore the gear between the motor and the wheel is not taken into account. The average efficiency of the powertrain is

$$\eta = \frac{P_o}{P_i} = \frac{\omega * \tau}{U_s * I_s} \quad (1)$$

, in which η is efficiency [-]
P_o is average power output [W]
P_i is average power input [W]
ω is average rotation speed of the drive shaft of the motor[rad/s]
τ is average torque [Nm]
U_s is average supply voltage [V]
I_s is average supply current [A].

5 Results

The Results chapter consists of two sections. In the first section the developed web application is presented. This comprehends the implementation and functionalities of the features defined in Table 2. The fulfilment of the requirement specification is discussed in Discussion 6. The second section presents the data acquired from testing and gives suggestions for possible education methods, which could be beneficial for students in order to learn mechatronics and development of IoT devices.

5.1 The EV control web application

This section of the results presents the final solutions to fulfill the required tasks. The web application browser view is shown in Figure 15. In the black section, number 1, the data provided by the vehicle via CAN bus is listed. Inside the red area, number 2, the user can set the maximum speed for the motor. The additional record and download features of the application are inside the blue, number 3, area.

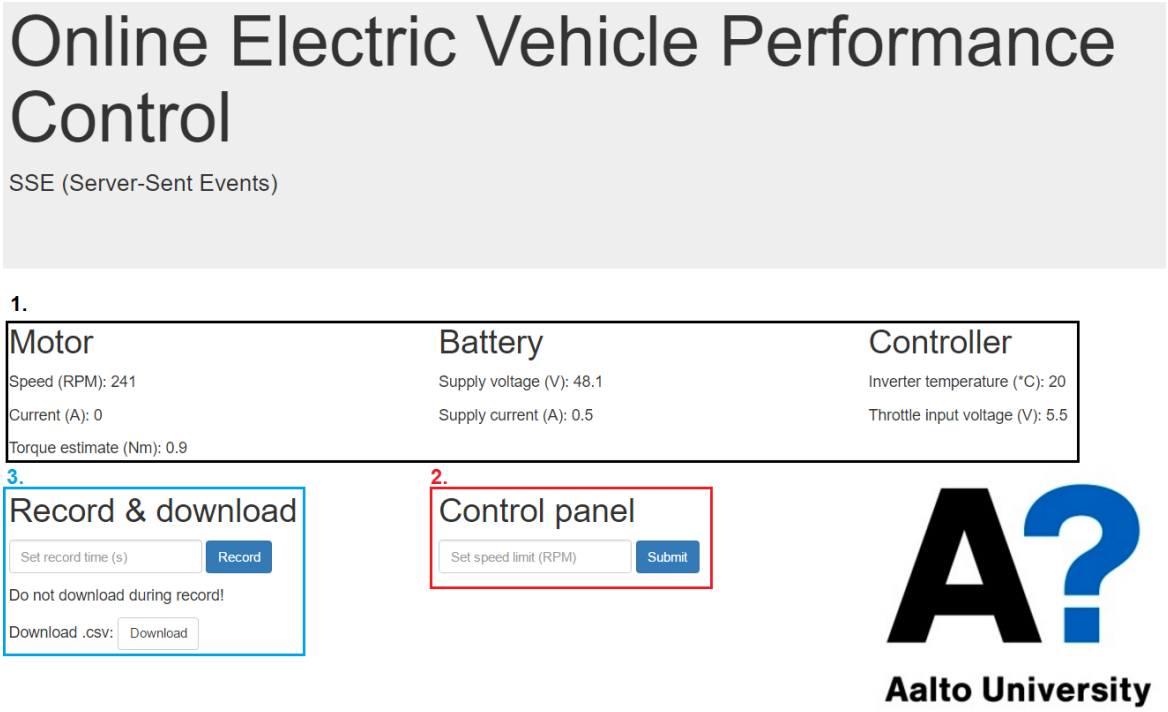

Figure 15 The browser view of the developed web application.

5.1.1 Data for client

The primary goal was to receive driving data from the vehicle and transmit the processed data to the client. The function developed to parse and process the CAN messages is in the script shown in appendix 1 (canthing.py) on lines 12-49. The function converts the message to string data format and collects the data in the message. Once current message is processed, the function returns the data as a python dictionary. To pass the information in the dictionary to the client, the function, in appendix 2 (main.py) on lines 41-48 encodes the dictionary to a JSON and sends it to the client. The client receives new events on the server with the event source on the HTML file. The implementation of the event source on the client side is shown in appendix 3 (index.html) on line 25. Once the client receives new event, it calls function

in appendix 3 on lines 185-201, which updates values attached to HTML elements. An example view of running data on the application page is shown in Figure 15 inside the black box, number 1.

5.1.2 Parameters to vehicle

The secondary goal was to send command to adjust the running process, in this case control the parameters of the ECU. With this feature, the user can control the electric vehicle instead of just monitoring the state of the process.

The aim was to develop a function which can be used to control the throttle of the vehicle but the properties of the ECU used prevents multiple simultaneous throttle inputs. The manual throttle was preserved and a feature to adjust the maximum rotation speed of the motor was developed.

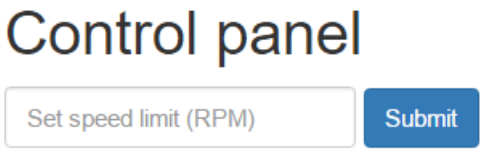

Figure 16 Control panel.

To adjust the parameter, the user utilizes the input field under the header "*Control panel*" in Figure 16 and submits the change. The input field and submit button are declared in the form element in appendix 3 on lines 111-120. The use of the button is detected by the client in the JavaScript function in appendix 3 on lines 170-181. This function uses HTTP post request method to communicate with the server. The performed request sends the maximum speed in the input field to the server as a part of the request message. The detection of the request on the server is shown in appendix 2 on line 60. Once this router notices a request, it initializes a function in appendix 2 on lines 61-67, which converts the posted value to hexadecimal format and sends the value as CAN message to the ECU of the vehicle. Once the message is sent to the controller, the server sends an appropriate response to the client.

5.1.3 Database control

To monitor and control the vehicle with the developed web application were the main and secondary goals. Nevertheless, these features are not enough, because the data is lost once displayed on the web application. To make the application more useful for teaching and research, a possibility to save and download a data file was added.

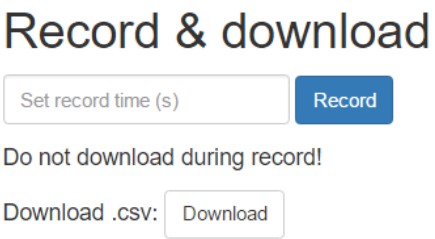

Figure 17 Record and download panel.

To initialize the data record, the user writes the desired time in seconds to the input field under the header "*Record & download*", shown in Figure 17, and submits the value with the record button. The input field and record button are declared in the form element in appendix

3 on lines 88-95. The JavaScript to detect the record initialization is shown in appendix 3 on line 155. In the same function, on line 163, the client sends a HTTP post request to the server with the record time as data to initialize the record. The request is detected on the server with a script shown in appendix 2 on line 50. When the request is noticed the function on lines 51-53 is called with the record time as an argument. This function is used to call a function ,in appendix 1 on lines 60-84, which creates a CSV file and writes the processed CAN bus data to the file. Once the data record is complete, the file created is closed and the server transmits an appropriate response to the client. Once the event to save the driving data to a file is initialized, the previous file saved in the server database is overwritten and lost.

Timestamp TPDO1 (s)	Supply voltage (V)	Supply current (A)	Speed (RPM)
1485245135.1206913	47.8125	0.875	396
1485245135.1498137	47.8125	0.875	399
1485245135.1803005	47.875	0.875	411
1485245135.2107053	47.875	0.875	402

Figure 18 Sample of the recorded file.

To ensure easy and convenient access to the created data file, a download button was implemented to the application. To download the data file from the server the user initializes the download event with the download button under the “Record & download” header, shown in Figure 17. The download button redirects the client to a new URL. The button implementation is shown in appendix 3 on lines 100-104. The href attribute in the anchor tag defines the requested URL, that is in this case the URL to initialize download. The detection of the request on the server is shown in appendix 2 on line 55. Once this router notices a request it initializes a function on lines 56-58 to respond to the request with a file from the server. A sample of the recorded data is shown in Figure 18.

5.2 Teaching methods

This section of the results presents the data acquired with the developed platform, and based on the acquired data, student exercises and project topics are created.

5.2.1 Correction of speed offset

Results of the maximum speed analysis show that the measured values of the maximum speed are smaller than the selected maximum speed. The average speeds for the maximum speed analysis are show in Table 8 and a graph based on the measurements in Figure 19.

Test #:	Target speed:	Measured speed:	Test #:	Target speed:	Measured speed:
1	100	92	11	600	564
2	150	139	12	650	612
3	200	186	13	700	660
4	250	233	14	750	706
5	300	281	15	800	755
6	350	328	16	850	802
7	400	374	17	900	849
8	450	422	18	950	896
9	500	469	19	1000	943
10	550	517	-	-	-

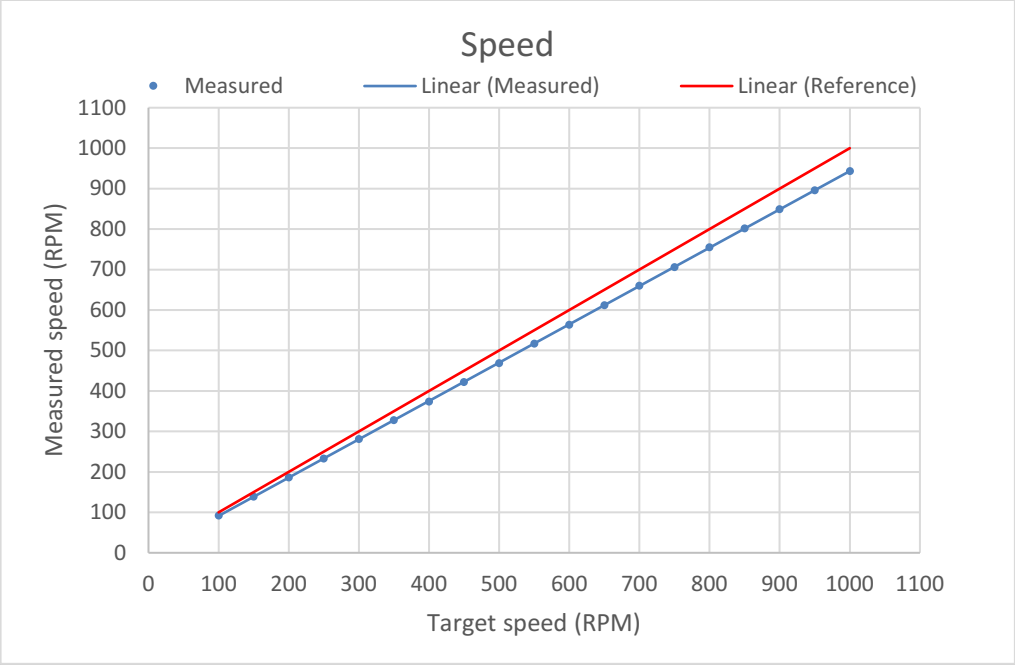

Figure 19 Average speeds without correction.

In Figure 19 the target speed on x-axis is the desired maximum rotation speed. The reference line has a slope of one and represents an ideal situation in which the measured speed corresponds to the selected maximum speed without an offset. The measured data points are the

values in Table 8. Excel equation tool was used to fit a trend line according to the measured data. The equation of the trend line is

$$N_{ms} = 0.9468 * N_{ts} - 3.5 \quad (2)$$

, in which N_{ms} is the measured speed [RPM] and N_{ts} the target speed [RPM]. The difference of the slopes of the reference line and the trend line is 0,0532. To reduce the difference the equation (3) is solved for target speed

$$N_{ts} = (N_{ms} + 3.5)/0.9468 \quad (3)$$

and the equation (4) is implemented to the *set_speed_limit* function in appendix 2 on line 62. After the implementation of the equation to the software, the speed test were performed again. The results are shown in Table 9 and the graph generated according to the data is in Figure 20.

Test #:	Target speed:	Measured speed:	Test #:	Target speed:	Measured speed:
1	100	101	11	600	601
2	150	151	12	650	649
3	200	201	13	700	698
4	250	251	14	750	749
5	300	301	15	800	800
6	350	350	16	850	850
7	400	400	17	900	899
8	450	449	18	950	950
9	500	501	19	1000	1000
10	550	550		-	-

Figure 20 Average speed with correction.

The blue linear trend line is fitted according to the new measured average speeds. From the equation of the trend line

$$N_{ms} = 0.9982 * N_{ts} + 1.0561 \quad (4)$$

the slope of the trend line is now 0.9982, which reduces the difference of the slopes of the reference line, in Figure 19, and the new trend line, in Figure 20, to 0.0018. In addition, the correction reduces the constant offset of 3.5 to 1.0561.

The green linear correction line in Figure 20 is calculated according to the equation (3) and represents the values to be set in order to achieve the desired maximum speed. For example if the wanted maximum speed is 500 RPM, the software calculates and transmits new value. 500 RPM results in a parameter value of 532 RPM.

The first teaching method is to repeat the measurements and to figure out a correction equation, which can be used to minimize the slope difference (1 – slope of a correction equation) and the constant offset.

The speed offset correction requires the students to operate the system without supervision. First the students gather the data without correction and based on the data the students generate a correction equation. After the required equation has been generated, it is implement to the platform software and the measurements are repeated. The final part of the exercise is to draw up a report including the measurement procedure, data without correction, suggested correction equation, corrected data and how significant the improvement was.

5.2.2 The efficiency problem

The efficiency of the system was estimated as described in section 4.1. The results of the efficiency measurement and calculation are shown in Table 10.

Table 8 Efficiency calculation results.

Supply voltage:	47.8 V
Supply current:	1.1 A
Input power:	52.6 W
Rotation speed:	52.5 rad/s
Torque:	1,1 Nm
Output power:	57.8 W
Efficiency:	1.1

The efficiency in Table 10 was calculated with equation (1). The efficiency of 110 % indicates that the power train of the vehicle would produce more mechanical energy than it consumes electrical energy, thus the power train would be a perpetual motion machine, which violates the first and second law of thermodynamics.

The efficiency problem project requires vast amount of independent work with the vehicle and the platform. The main problem most probably lies within the torque estimate provided by the ECU. Therefore, the values used to estimate the efficiency requires verification, which can be challenging and time consuming.

The second teaching method is to solve what causes the problem in the powertrain and how it could be fix. As a final part, the solution is implemented to the platform to improve the quality of the data.

5.2.3 IoT-demo device

The third way to utilize the platform in teaching is as a demo device. In the demo, students will learn how web application development tools could be used to send data to and from their own developed systems.

The scale of the IoT-demo indicates that the needed tools and technologies can be presented to students, i.e., the learning goal can be achieved approximately in 2-4 hours depending on how thorough the demo is. In this teaching method the students do not utilize the platform on their own but the demo host shows different features and the implementation explaining how they are fulfilled.

Table 9 Suggested teaching methods.

Method name:	Learning goal:	Scale:
IoT-demo	To know how to connect a device to the internet	Demo
Speed offset correction	Improve system and/or process with data analysis	Exercise
Efficiency problem	How to verify data and understand how an EV works	Project

6 Discussion

The objective of this study was to support the aim of the A!OLE project in terms of developing and improving technical solutions to support learning and the teaching of mechatronic devices associated with internet. The measures taken to achieve the goal was the design and development of an internet connected mechatronic machine which is to be utilized in education.

The outcome was a light weight electric vehicle which can be monitored and controlled with a web application. In addition, different types of teaching methods were created. Both the vehicle and the control application are used for these teaching methods. Apart from implementing a visual presentation, a graph, of the acquired data, all requirements imposed in Table 2 were fulfilled. The lack of this feature on the web application does not restrict the usage of the developed system in teaching.

The main differences of the teaching methods presented in Table 11 are scale and learning goals. The scale of the methods is described with the terms demo, exercise and project which of each indicate how time consuming and intense the learning is. The demo takes least amount of time with the smallest intensity, exercise can be performed in a week or less with minor intensity and project requires much more time and intense work.

Two of the three created teaching methods presented in section 5.2 Teaching methods are based on the observations in the system. The first inconsistency concerns the efficiency problem that results from faulty data. The reason could be that the torque estimate provided by the electrical controller unit is wrong. The controller does not measure the torque but estimates it most probably by multiplying the supply current with a scaling factor. It must be considered that the controller is designed to support 48 V system, which might lead to a wrong torque estimate received while driving a motor with nominal voltage of 96 V. The second problem is the offset between set maximum rotation speed and the measured speed. The controller might have a safety factor to prevent the motor from achieving the set maximum speed. The aim of the exercise correcting the offset is not to study how electronic controller units work, but to generate a mathematical model based on the measurement data to fix a problem. The teaching methods are dissimilar and therefore improve different skills needed in engineering. The usage of the designed system is not restricted solely to be used with these exercises and it is advised to develop the presented methods and to create new methods based on student feedback.

The system development was restricted to enable full-duplex communication between the vehicle and the client. Therefore, quality inspection was not performed for the features implemented to the application. The data transfer time from vehicle to client was not measured, thus it is impossible to evaluate if the data transmitted by the controller is displayed in real time to the user. In addition, the developed software does not support saving multiple data files on the server at the same time. To perform multiple measurements, the file must be acquired before the next test, which makes performing multiple measurements in series time consuming and inconvenient.

The system is sufficient as a teaching device, but in order to benefit researchers further development is needed. The first recommendation is to implement GPS location tracking and inclination angle measurement. With these improvements, the terrain profile could be recorded and for example the effects of hills to the efficiency of the powertrain could be studied.

The second recommendation is to make the application available in World Wide Web. This would enable global measurements and data acquisitions. The challenges with internet security must be handled before the application is made available in the World Wide Web.

Bibliography

[1] D. Bradley, D. Russell, I. Ferguson, J. Isaacs, A. Macleod, and R. White. The Internet of Things - The future or the end of mechatronics. *Mechatronics*. [Online magazine]. Vol. 27, pp. 57–74, 2015. [Accessed: 10-Feb-2017]. Available: http://www.sciencedirect.com/science/article/pii/S0957415815000215. ISSN 09574 158 [Online]

[2] About. *A!OLE - Aalto Online learning*. 2016. [Online]. Available: http://onlinelearning.aalto.fi/about/. [Accessed: 23-Sep-2016].

[3] ISO 11898-1. Road vehicles - Controller area network (CAN) - Part 1: Data link layer and physical signalling. [Online document] Geneva: ISO. pp. 65., [Accessed: 15-Dec-2015]

[4] ISO 11898-2. Road vehicles - Controller area network (CAN) - Part 2: High-speed medium access unit. [Online document] Geneva: ISO. pp. 30 [Accessed: 1.12.2013]

[5] S. Corrigan. Introduction to the controller area network (CAN). [Online document]. Texas Instruments. Dallas, USA, 2002. [Accessed: 10-Dec-2016] Available: http://www.ti.com/lit/an/sloa101a/sloa101a.pdf

[6] W. Voss. *A Comprehensible Guide to Controller Area Network*. 2nd ed. Greenfield, Massachusetts, USA: Copperhill Media Corporation, 2008 pp. 152. ISBN 978-09765 11601

[7] Introduction to CAN. *Renesas*. 2006. [Online]. Available: https://www.renesas.com/enus/doc/products/mpumcu/apn/001/rej05b0804_m16cap.pdf. [Accessed: 10-Nov-2016].

[8] Controller Area Network (CAN) Overview. *National Instruments*. 2014. [Online]. Available: http://www.ni.com/white-paper/2732/en/. [Accessed: 01-Nov-2016].

[9] L. Wolfhard, *CAN System Engineering*, 2nd ed. Wolfenbüttel, Germany: Springer, 2013 pp. 353. ISBN 978-1-4471-5613-0 [eBook]

[10] CANopen - The standardized embedded network. *CAN in Automation (CiA)*. 2016. [Online]. Available: http://www.can-cia.org/can-knowledge/canopen/canopen/. [Accessed: 03-Nov-2016].

[11] CANopen Basics - Introduction. *CANopenSolutions*. 2016. [Online]. Available: http://www.canopensolutions.com/english/about_canopen/about_canopen.shtml. [Accessed: 03-Nov-2016].

[12] CiA Draft Standard 301. CANopen application layer and communication profile. [Online document] Nuremberg: CiA. 2016. Available: http://overpof.free.fr/schneider/CAN&CANopen/CANopen/%A9CiACANCANopenCD V5.1/standard/ds301.pdf. [Accessed: 13-Feb-2002].

[13] The Basics of CANopen. *National Instruments*. 2013. [Online]. Available: http://www.ni.com/white-paper/14162/en/. [Accessed: 03-Nov-2016].

[14] CANopen Basics - Communication. *CANopenSolutions*. 2016. [Online]. Available: http://www.canopensolutions.com/english/about_canopen/communication.shtml. [Accessed: 08-Nov-2016].

[15] CANopen Basics - Process Data Exchange. *CANopenSolutions*. 2016. [Online]. Available: http://www.canopensolutions.com/english/about_canopen/pdo.shtml. [Accessed: 03-Nov-2016].

[16] S. Fowler and V. Stanwick. *Interactive Technologies : Web Application Design Handbook : Best Practices for Web-Based Software*. San Francisco, California, USA: Morgan Kaufmann, 2004, pp. 689. ISBN 9780080481708 [eBook]. ISBN 9781558607521 [Print]

[17] A. Cooper and R. Reimann. *About Face 2.0: The Essentials of Interaction Design*. 2nd ed. Indianapolis, Indiana, USA: Wiley Publishing, Inc., 2003, pp. 540. ISBN 0-645-26413

[18] M. M. Alani. *Guide to OSI and TCP/IP Models*. London, UK: Springer, 2014, pp. 50. ISBN 978-3-319-05152-9 [eBook]. ISBN 978-3-319-05151-2 [Print]

[19] Hypertext Transfer Protocol. *Wikipedia*. 2017. [Online]. Available: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol. [Accessed: 09-Feb-2017].

[20] HTTP Methods: GET vs. POST. *W3Schools*. 2017. [Online]. Available: http://www.w3schools.com/Tags/ref_httpmethods.asp. [Accessed: 09-Jan-2017].

[21] D. Gourley, B. Totty, M. Sayer, A. Aggarwal, and S. Reddy. *HTTP: The Definitive Guide*. Sebastopol, California: O'Reilly Media, Inc., 2002, pp. 658. ISBN 978-1-56592-509-0

[22] I. Hickson. W3C Proposal Recommendation Server-Sent Events. 2014. [Online]. Available: https://www.w3.org/TR/2014/PR-eventsource-20141209/. [Accessed: 25-Jan-2017].

[23] E. Bidelman. Stream Updates with Server-Sent Events. *HTML5Rocks*. 2010. [Online]. Available: https://www.html5rocks.com/en/tutorials/eventsource/basics/. [Accessed: 25-Jan-2017].

[24] Server-sent events. *Mozilla developer network*. 2017. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events. [Accessed: 25-Jan-2017].

[25] HTML5 Server-Sent Events. *W3Schools*. 2016. [Online]. Available: http://www.w3schools.com/html/html5_serversentevents.asp. [Accessed: 25-Jan-2017].

[26] ECU Designing and Testing using National Instruments Products. *National Instruments*. 2009. [Online]. Available: http://www.ni.com/white-paper/3312/en/. [Accessed: 04-Nov-2016].

[27] Gen4 Applications Reference Manual Document no : 177/52701. *Sevcon*. 2009. [Online]. Available: http://www.thunderstruck-ev.com/Manuals/Gen4_Product_Manual_V3.0.pdf. [Accessed: 14-Dec-2016].

[28] Dynamic test. *Golden Motor*. 2013. [Online]. Available: http://www.goldenmotor.com/eCar/HPM96-10000.pdf. [Accessed: 04-Nov-2016].

[29] Motor test curve. *Golden Motor*. 2013. [Online]. Available: http://www.goldenmotor.com/eCar/HPM96-10000.pdf. [Accessed: 04-Nov-2016].

[30] E. Upton and G. Halfacree. *Raspberry Pi User Guide*. West Sussex, UK: Wiley, 2014, pp. 314. ISBN 9781118795460 [eBook]. ISBN 9781118795484 [Print]

[31] FAQs. *Raspberry Pi Foundation*. 2016. [Online]. Available: https://www.raspberrypi.org/help/faqs/#introWhatIs. [Accessed: 08-Nov-2016].

[32] Raspberry Pi 3 Model B datasheet. *Raspberry Pi Foundation*. 2016. [Online]. Available: http://docs-europe.electrocomponents.com/webdocs/1521/0900766b81521bab.pdf. [Accessed: 13-Dec-2016].

[33] PiCAN 2 User Guide. *SK Pang Electronics Ltd*. 2016. [Online]. Available: http://skpang.co.uk/catalog/images/raspberrypi/pi_2/PICAN2UG12.pdf. [Accessed: 13-Dec-2016].

[34] MCP2515 data sheet. *Microchip*. 2016. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/20001801H.pdf. [Accessed: 14-Dec-2016].

[35] MCP2551 data sheet. *Microchip*. 2010. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/21667f.pdf. [Accessed: 14-Dec-2016].

[36] A Brief Introduction to the Serial Peripheral Interface (SPI). *Arduino*. 2016. [Online]. Available: https://www.arduino.cc/en/Reference/SPI. [Accessed: 14-Dec-2016].

[37] Using DVT with Gen4 Systems. *Sevcon*. [Online]. Available: https://www.thundertruck-ev.com/images/DVTTutorial.pdf. [Accessed: 17-Jan-2016].

[38] What is Python? Executive Summary. *Python*. 2017. [Online]. Available: https://www.python.org/doc/essays/blurb/. [Accessed: 09-Feb-2017].

[39] M. Telles. *Python, Power!*. Bosto, Massachusetts, USA: Course Technology, 2007, pp. 529. ISBN: 9781598631593 [eBook]. ISBN 9781598631586 [Print]

[40] Applications for Python. *Python*. 2017. [Online]. Available: https://www.python.org/about/apps/. [Accessed: 09-Feb-2017].

[41] About JavaScript. *Mozilla developer network*. 2017. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript. [Accessed: 09-Feb-2017].

[42] D. S. McFarland. *JavaScript & jQuery: The Missing Manual*. 2nd ed. Sebastopol, California, USA: O'Reilly Media, Inc., 2011, pp. 538. ISBN 978-1-4493-9902-3

[43] First-class function. *Wikipedia*. 2017. [Online]. Available: https://en.wikipedia.org/wiki/First-class_function. [Accessed: 09-Feb-2017].

[44] jQuery. *jQuery*. 2017. [Online]. Available: https://jquery.com/. [Accessed: 09-Feb-2017].

[45] HTML. *Wikipedia*. 2017. [Online]. Available: https://fi.wikipedia.org/wiki/HTML. [Accessed: 09-Feb-2017].

[46] Introduction to HTML. *Mozilla developer network*. 2017. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML. [Accessed: 09-Feb-2017].

[47] Cascading Style Sheets. *Wikipedia*. 2017. [Online]. Available: https://en.wikipedia.org/wiki/Cascading_Style_Sheets. [Accessed: 09-Feb-2017].

[48] J. M. Gustafson. *HTML5 Web Application Development by Example*. Birmingham, UK: Packt Publishing, 2013, pp. 371. ISBN 9781849695954 [eBook]. ISBN 9781849695947 [Print]

[49] PiCAN2 CAN-Bus Board for Raspberry Pi 2/3. *SK Pang Electronics*. 2016. [Online]. Available: http://skpang.co.uk/catalog/pican2-canbus-board-for-raspberry-pi-23-p-1475.html. [Accessed: 03-Jan-2017].

[50] B. Powell. Python-CAN libraries. *BitBucket*. 2007. [Online]. Available: https://bitbucket.org/hardbyte/python-can/get/4085cffd2519.zip. [Accessed: 03-Jan-2017].

[51] Werkzeug. *Pocoo*. 2016. [Online]. Available: http://werkzeug.pocoo.org/docs/0.11/#. [Accessed: 03-Jan-2017].

[52] Jinja2. *Pocoo*. 2016. [Online]. Available: http://jinja.pocoo.org/docs/dev/. [Accessed: 03-Jan-2016].

[53] Flask. *Pocoo*. 2010. [Online]. Available: http://flask.pocoo.org/. [Accessed: 03-Jan - 2017].

[54] Web Server Gateway Interface. *Wikipedia*. 2016. [Online]. Available: https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface. [Accessed: 16-Feb-2017].

[55] R. DuPlain. *Instant Flask Web Development*. Birmingham, UK: Packt Publishing, 2013, pp. 85. ISBN 9781782169635 [eBook]. ISBN 9781782169628 [Print]

List of appendixes

Appendix 1. Python: canthing.py

```python
import threading
import time

class CanThing(object):
    """Internet "thing" that can read and send CANopen bus on RasPi."""

    def __init__(self):
        """Decler threading lock to prevent ununiform
            data distribution while recording"""
        self._lock = threading.Lock()

    def __read_messages(self, bus, TPDO, sf):
        for msg in bus:
            msg = str(msg)
            cod_id = msg[25:29]
            timestamp = msg[0:17]
            if cod_id == TPDO["TPDO1_id"]:
                RPM = int(msg[54:56] + msg[51:53] + msg[48:50] +
                          msg[45:47], 16)*sf["motor_RPM"]
                if RPM >= 20000:
                    RPM = 0
                battery_V = int(msg[60:62] + msg[57:59], 16)
                                *sf["battery_V"]
                battery_C = int(msg[66:68] + msg[63:65], 16)
                                *sf["battery_C"]
                if battery_C >= 4095:
                    battery_C = 0
                return {"cod_id": cod_id,
                        "rpm": RPM,
                        "battery_V": battery_V,
                        "battery_C": battery_C,
                        }

            elif cod_id == TPDO["TPDO2_id"]:
                motor_C = int(msg[48:50] + msg[45:47], 16)*sf["motor_C"]
                inverter_temp = int(msg[51:53], 16)*sf["inverter_T"]
                motor_torque = int(msg[57:59] + msg[54:56], 16)
                                   *sf["motor_torque"]
                motor_torque = (motor_torque * 36.2)/100
                if motor_torque >= 3270:
                    motor_torque = 0
                throttle_V = int(msg[63:65] + msg[60:62], 16)
                                 *sf["throttle_V"]
                return {"cod_id": cod_id,
                        "motor_C": motor_C,
                        "inverter_temp": inverter_temp,
                        "motor_torque": motor_torque,
                        "throttle_V": throttle_V
                        }

```

```
51      def __msg_to_thing(self, bus, TPDO, sf):
52          """lock other threads and release lock
53            when other functions called"""
54          with self._lock:
55              return self.__read_messages(bus, TPDO, sf)
56
57      def __msg_to_CSV(self, bus, TPDO, sf):
58          return self.__read_messages(bus, TPDO, sf)
59
60      def write_CSV(self, bus, TPDO, sf, record_time):
61          """locks other threads during .csv file writing
62            and releases lock when ready"""
63          with self._lock:
64              file = open("data.csv", "w")
65              file.write("Timestamp TPDO1 (s), Supply voltage (V),"
66                         "Supply current (A), Speed (RPM),,"
67                         "Timestamp TPDO2 (s), Torque (Nm),"
68                         "Motor Current(A), Inverter temperature (*C),"
69                         "Throttle input voltage (V)\n")
70              time_start = time.time()
71              while time.time() < time_start + record_time:
72                  msg = self.__msg_to_CSV(bus, TPDO, sf)
73                  if msg["cod_id"] == TPDO["TPDO1_id"]:
74                      file.write(str(time.time()) + "," +
75                                 str(msg["battery_V"]) + "," +
76                                 str(msg["battery_C"]) + "," +
77                                 str(msg["rpm"]) + ",")
78                  elif msg["cod_id"] == TPDO["TPDO2_id"]:
79                      file.write("," + str(time.time()) + "," +
80                                 str(msg["motor_torque"]) + "," +
81                                 str(msg["motor_C"]) + "," +
82                                 str(msg["inverter_temp"]) + "," +
83                                 str(msg["throttle_V"]) + "\n")
84              file.close()
```

Appendix 2. Python: main.py.

```python
from flask import *
from canthing import CanThing
import os
import can
import json
import csv

#Dictionaries for TPDO ids and scaling factors (sf)

TPDO = {"TPDO1_id": "0100",
        "TPDO2_id": "0200",
        "TPDO3_id": "0300", #SEVCON not transmitting & no binded data
        "TPDO4_id": "0400", #SEVCON not transmitting & no binded data
        "TPDO5_id": "0500" #SEVCON not transmitting & no binded data
        }

sf = {"motor_RPM": 1,
      "motor_C": 1,
      "inverter_T": 1,
      "battery_V": 0.0625,
      "battery_C": 0.0625,
      "motor_torque": 0.1,
      "throttle_V": 0.00390625
      }

# Bring CAN interface up (1Mb/s):
os.system("sudo /sbin/ip link set can0 up type can bitrate 1000000")
bus = can.interface.Bus(channel='can0', bustype='socketcan_native')

can_thing = CanThing()
app = Flask(__name__)

# Define app routes
# Index route renders the main HTML page
@app.route("/")
def index():
    return render_template("index.html")

# Server-sent event endpoint that streams the thing data
@app.route("/thing")
def thing():
    def read_data():
        while True:
            # Read message and save data as dictionary
            msg = can_thing._CanThing__msg_to_thing(bus, TPDO, sf)
            # Send thing data (/templates/index.html) as JSON object
            yield("data: {0}\n\n".format(json.dumps(msg)))
    return Response(read_data(), mimetype="text/event-stream")

```

```
50  @app.route("/record/<int:record_time>", methods = ["POST"])
51  def record(record_time):
52      can_thing.write_CSV(bus, TPDO, sf, record_time)
53      return ("", 204)
54
55  @app.route("/download")
56  def download():
57      return send_from_directory("/home/pi/programs/webapp_SSE_A",
58                                 "data.csv", as_attachment = True)
59
60  @app.route("/set_speed_limit/<int:speed_limit>", methods =["POST"])
61  def set_speed_limit(speed_limit):
62      speed_limit = round(int((speed_limit + 3.5)/0.9468))
63      byte2 = speed_limit//255
64      byte1 = speed_limit - byte2*255
65      msg = can.Message(arbitration_id=0x200 ,data=[byte1, byte2])
66      bus.send(msg)
67      return ("", 204)
68
69  # Start the flask debug server listening on pi port 5000 as default
70  if __name__ == "__main__":
71      app.run(host="0.0.0.0", debug = True, threaded = True)
```

Appendix 3. HTML & JS: index.html

```
<!doctype html>
<html class="no-js" lang="">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="x-ua-compatible" content="ie=edge">
        <title>Akkuhuone</title>
        <meta name="description" content="">
        <meta name="viewport" content="width=device-width,
        initial-scale=1">
        <link rel="stylesheet" href="{{ url_for('static',
        filename='css/bootstrap.min.css') }}">
        <link rel="shortcut icon"  href="{{url_for('static',
        filename='favicon.ico') }}">
    </head>
    <body>
        <!--[if lt IE 8]>
            <p class="browserupgrade">You are using an
            <strong>outdated</strong> browser. Please
            <a href="http://browsehappy.com/">upgrade your browser</a>
            to improve your experience.</p>
        <![endif]-->
        <!-- Add your site or application content here -->

        <div class="jumbotron">
            <div class="container">
                <h1>
                Online Electric Vehicle Performance Control
                </h1>
                <p>
                SSE (Server-Sent Events)
                </p>
            </div>
        </div>

```

```
35      <div class="container">
36          <div class="row">
37              <div class="col-md-4">
38                  <h2>
39                      Motor
40                  </h2>
41                  <p>
42                      Speed (RPM): <span id="rpm_val">
43                      </span>
44                  </p>
45                  <p>
46                      Current (A): <span id="motor_C_val">
47                      </span>
48                  </p>
49                  <p>
50                      Torque estimate (Nm): <span
51                      id="motor_torque_val"></span>
52                  </p>
53              </div>
54
55              <div class="col-md-4">
56                  <h2>
57                      Battery
58                  </h2>
59                  <p>
60                      Supply voltage (V): <span id="battery_V_val">
61                      </span>
62                  </p>
63                  <p>
64                      Supply current (A): <span id="battery_C_val">
65                      </span>
66                  </p>
67              </div>
68              <div class="col-md-4">
69                  <h2>
70                      Controller
71                  </h2>
72                  <p>
73                      Inverter temperature (*C): <span
74                      id="inverter_temp_val"></span>
75                  </p>
76                  <p>
77                      Throttle input voltage (V): <span
78                      id="thottle_V_val"></span>
79                  </p>
80              </div>
81          </div>
```

```
82          <div class="row">
83              <div class="col-md-4">
84                  <h2>
85                      Record & download
86                  </h2>
87                  <p>
88                  <form class="form-inline">
89                  <div class="form-group">
90                  <input type="val" class="form-control input-sm"
91                  id="record_time" placeholder="Set record time (s)">
92                  </div>
93                  <button type="submit" class="btn btn-primary btn-sm"
94                  id="record_data">Record</button>
95                  </form>
96                  </p>
97                  <p>
98                      Do not download during record!
99                  </p>
100                 <p>
101                 Download .csv: <a class="btn btn-default btn-sm"
102                 id="dwn_btn" href = "/download" type="button">
103                 Download</a>
104                 </p>
105             </div>
106             <div class="col-md-4">
107                 <h2>
108                     Control panel
109                 </h2>
110                 <p>
111                 <form class="form-inline">
112                 <div class="form-group">
113                 <input type="val" class="form-control input-sm"
114                 id="max_speed_val"
115                 placeholder="Set speed limit (RPM)">
116                 </div>
117                 <button type="submit"
118                 class="btn btn-primary btn-sm"
119                 id="set_max_speed_val">Submit</button>
120                 </form>
121                 </p>
122             </div>
123
124             <div class="col-md-4">
125                 <p>
126                 <img src = "/static/aalto_logo.jpg" alt="140x140"
127                 class = "img-rounded">
128                 </p>
129             </div>
130         </div>
131     </div>
```

```
132
133         <script src="https://code.jquery.com/jquery-1.12.0.min.js">
134         </script>
135         <script>window.jQuery || document.write(
136         '<script src="js/vendor/jquery-1.12.0.min.js"><\/script>')
137         </script>
138         <script
139         src="{{ url_for('static', filename='js/bootstrap.min.js') }}">
140         </script>
141
142         <script>
143             //Runs once the page Document Object Model (DOM) is ready
144             //for JavaScript code to execute. A page can't be
145             //manipulated safely until the document is "ready." jQuery
146             //detects this state of readiness.
147             $(document).ready(function() {
148                 var initial_max_speed = 255;
149                 $.post("/set_speed_limit/"+initial_max_speed);
150             });
151
152             // Record data. On button click (id = "record_data")
153             // record function initialized
154             $(function() {
155                 $("#record_data").click(function() {
156                     //Save input id="record_time" to variable time
157                     var time = $("#record_time").val();
158                     if(isNaN(time) || time === "" || time == " ") {
159                         console.log("Input NaN!");
160                     }else{
161                         //HTTP request (POST) to start record function
162                         //on main.py wtih time argument
163                         $.post("/record/"+time);
164                     }
165                     return false;
166                 });
167             });
168
169             // Set speed limit.
170             $(function() {
171                 $("#set_max_speed_val").click(function() {
172                     var max_speed = $("#max_speed_val").val();
173                     if(isNaN(max_speed) || max_speed === "" ||
174                         max_speed == " ") {
175                         console.log("Input NaN!");
176                     }else{
177                         $.post("/set_speed_limit/"+max_speed);
178                     }
179                     return false;
180                 });
181             });
```

```
182
183             //Function to update values.
184             //Extend if-elif structure if more TPDOs added
185             function updateData(msg) {
186                 if (msg.cod_id === "0100") {
187                     $("#rpm_val").text(msg.rpm);
188                     $("#battery_V_val").text(Math.round(
189                         msg.battery_V*10)/10);
190                     $("#battery_C_val").text(Math.round(
191                         msg.battery_C*10)/10);
192                 }
193                 else if (msg.cod_id === "0200") {
194                     $("#motor_C_val").text(msg.motor_C);
195                     $("#inverter_temp_val").text(msg.inverter_temp);
196                     $("#motor_torque_val").text(Math.round(
197                         msg.motor_torque*10)/10);
198                     $("#thottle_V_val").text(Math.round(
199                         msg.throttle_V*10)/10);
200                 }
201             }
202
203             //Setup thing data server sent event receiver.
204             //Receives data from function thing (def thing) on main.py
205             var thingSource = new EventSource("{{url_for("thing")}}");
206             //Initialized once new message received
207             thingSource.onmessage = function(e) {
208                 //Call updateData with parsed json object
209                 updateData($.parseJSON(e.data));
210                 }
211         </script>
212     </body>
213 </html>
```